HALBORN

Rarimo - Snap App

WebApp Pentest

Prepared by: **Halborn**

Date of Engagement: **July 24th, 2023 - July 27th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	07/24/2023	Carlos Polop
0.2	Draft Review	07/27/2023	Gabi Urrutia
1.0	Remediation Plan	08/18/2023	Carlos Polop
1.1	Remediation Plan Review	08/18/2023	Gabi Urrutia
1.2	Remediation Plan Update	08/24/2023	Carlos Polop
1.3	Remediation Plan Update Review	08/28/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
carlos.polop	Halborn	carlos.polop@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Rarimo engaged Halborn to conduct a security assessment on their custom pre-released software Metamask snap beginning on July 24th, 2023 and ending on July 27th, 2023. The security assessment was scoped to the forked Metamask snap repository to the Halborn team.

1.2 ASSESSMENT SUMMARY

The team at Halborn was provided one week for the engagement and assigned a full-time security engineer to verify the security of the application. The security engineer is a penetration testing expert with advanced knowledge in web, recon, discovery & infrastructure penetration testing.

The purpose of this assessment is to ensure that the Metamask snap application is secure and cannot be unexpectedly abused to get access to other users' assets.

After a careful review of the code and a dynamic analysis of the snap application, it was determined that the attack surface was highly limited as only 5 functions were being exposed by the snap application: `create_identity`, `save_credentials`, `create_proof`, `create_backup`, `recover_backup`.

UPDATE: After the update of the code another function was added and exposed, the function `CheckStateContractSync`, making the attack surface a bit bigger. However, the logic of the new function is very limited, not allowing to an attacker to control any part of the flow of the function.

Although those functions allow performing some privileged actions, Metamask prompts the user to authorize the installation of the Rarimo's snap application and to authorize webs to be able to communicate with the snap application. Therefore, only pre-authorized websites by the user will be able to contact the snap application.

Overall, a couple of medium vulnerabilities were found related to the

trust of unsanitized input, also a low vulnerability regarding lack of pinned versions and an info vulnerability to improve the security of the Snap App were reported.

UPDATE: After reviewing the changes, no new issues have been found in the application.

In summary, Halborn identified some security risks that were mostly addressed by the Rarimo team.

1.3 SCOPE

The analyzed snap application was https://gitlab.com/rarimo/identity/metamask-snap/ with the commit id 960285ece82bb2fe907813e0c8bb31a82e9915bb (the latest commit in the main branch at the starting date of this assessment).

UPDATE: The code was updated to the commit 6efa918aeb21d7c5e154e20b048754f417ce0f16 with the following changes:

- Added a method for checking the state
- Changed the method of proof generation
- Changed the method of creating identity
- https://gitlab.com/rarimo/identity/metamask-snap/-/merge_requests/9/diffs
- https://gitlab.com/rarimo/identity/metamask-snap/-/merge_requests/10/diffs

1.4 TEST APPROACH & METHODOLOGY

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities.

The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	1	1

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) POTENTIAL SAVE OF ARBITRARY CREDENTIALS	Medium	SOLVED - 08/18/2023
(HAL-02) POTENTIAL GENERATION OF ARBITRARY PROOFS	Medium	SOLVED - 08/18/2023
(HAL-03) DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS	Low	SOLVED - 08/18/2023
(HAL-04) RESTRICT SITES	Informational	ACKNOWLEDGED

FINDINGS & TECH DETAILS

3.1 (HAL-01) POTENTIAL SAVE OF ARBITRARY CREDENTIALS - MEDIUM

Description:

The lack of proper input sanitization can lead to several security vulnerabilities in a software system. An attacker could exploit unsanitized inputs to inject malicious code or commands, potentially leading to unauthorized data access, modification, or deletion; circumvention of security measures; arbitrary code execution; and overall system compromise.

Details:

During the assessment, it was discovered that the exposed RPC endpoint `save_credentials` is not checking or sanitizing the data received:

Listing 1

```
1 const offer = request.params as any as ClaimOffer;
```

This data is later used in the rest of the code of the RPC:

Listing 2

```
1  const dialogCredentials = offer.body.credentials.reduce(
2      (acc: any, cred: any) => {
3        return acc.concat([divider(), text(cred.description), text(
↳ cred.id)]);
4      },
5      [],
6  );
7
8  const res = await snap.request({
9      method: 'snap_dialog',
10     params: {
11       type: 'confirmation',
12       content: panel([...dialogContent, ...dialogCredentials]),
13     },
```

```
14 });
15
16 if (res) {
17     const identity = await Identity.create(identityStorage.
↳ privateKeyHex);
18     const authProof = new AuthZkp(identity, offer);
19     const credentials = await authProof.getVerifiableCredentials()
↳ ;
20     await saveCredentials(credentials);
21     return credentials;
22 }
```

An attacker with access to execute JS code in the context of a page authorized to contact this Snap App (for example via a Cross-Site Scripting vulnerability) could invoke this endpoint and send arbitrary credentials, partially controlling the flow that uses this data, and finally storing arbitrary credentials.

Risk Level:

Likelihood - 3
Impact - 3

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:L

Recommendation:

It is recommended to check that each value in the parameters received contains the expected format and that it was legitimately sent by the user.

Note that the criticality of this vulnerability was reduced because it requires user interaction. However, note how the attacker would be controlling the `offer.from` string parameter and other string parameters used to ask the user to allow saving the credentials.

Another specially sensitive parameter the attacker could control is `url`.

As the parameters sent by the potential attacker are going to be stored in an object of type `ClaimOffer`, it is recommended to check each of the fields of this object with regexes to ensure their format (and potentially compare the URL with a list of allowed domains to use):

Listing 3

```
export type ClaimOffer = {
  body: {
    credentials: [
      {
        description: string;
        id: string;
      },
    ];
    url: string;
  };
  from: string;
  id: string;
  thid?: string;
  to: string;
  typ?: string;
  type: string;
};
```

Another option to consider fixing this issue would be to add all the logic the client JavaScript is executing (the `create_identity` call and the fetch request) before calling `save_credentials` to the `save_credentials` code, so the potential attacker will not be able to send any data.

Remediation Plan:

SOLVED: The function `isValidSaveCredentialsOfferRequest` was added to validate the `offer` before using it.

3.2 (HAL-02) POTENTIAL GENERATION OF ARBITRARY PROOFS - MEDIUM

Description:

Similar to the previous issue, this vulnerability is related to the lack of sanitization of the received parameters that allows a potential attacker with access to communicate with the Snap App to try to create proofs

Details:

In this case, it was observed that a potential attacker could abuse, for example, a XSS vulnerability to generate an arbitrary proof.
This is because the attacker would be in complete control of the parameters used to create the proof because there is no input sanitization.

The parameters are loaded in the line:

Listing 4

```
1 const params = request.params as any as CreateProofRequest;
```

And later used to generate the proof without any checking or sanitization:

Listing 5

```
1 const credentials = (await findCredentialsByQuery(params.query)).
↳ filter(
2     (cred) => cred.credentialSubject.id === identityStorage.did,
3 );
4
5 if (!credentials.length) {
6     throw new Error(
7       `no credential were issued on the given id ${identityStorage
↳ .did}`,
8     );
9 }
```

```
10
11 const credentialType = params.query.type;
12 const { credentialSubject } = params.query;
13 const { circuitId } = params;
14
15 const res = await snap.request({
16     method: 'snap_dialog',
17     params: {
18       type: 'confirmation',
19       content: panel([
20         heading('Create proof'),
21         ...(credentialType
22           ? [divider(), text('Credential type'), text(
↳ credentialType)]
23           : []),
24         ...(credentialSubject
25           ? [
26               divider(),
27               text('Requirements'),
28               ...Object.keys(credentialSubject).reduce(
29                 (acc: TextField[], fieldName) => {
30                   const fieldOperators = credentialSubject?.[
↳ fieldName];
31                   const textField = Object.keys(fieldOperators).
↳ map(
32                     (operator) => {
33                       return text(
34                         `${fieldName} - ${operator} ${
↳ fieldOperators[operator]}\n`,
35                       );
36                     },
37                   );
38                   return acc.concat(textField);
39                 },
40                 [],
41               ),
42             ]
43           : []),
44         ...(circuitId
45           ? [divider(), text('Proof type'), text(circuitId)]
46           : []),
47       ]),
48     },
49 });
```

```
50
51 if (res) {
52     const identity = await Identity.create(identityStorage.
↳ privateKeyHex);
53     const zkpGen = new ZkpGen(identity, params, credentials[0]);
54     return await zkpGen.generateProof();
55 }
```

Risk Level:

Likelihood - 3
Impact - 3

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:U/C:L/I:L/A:L

Recommendation:

It is recommended to check that each value in the parameters received contains the expected format and that it was legitimately sent by the user.

Note that the criticality of this vulnerability was reduced because it requires user interaction. However, note how the attacker would be controlling the parameters used to ask the user to allow generating the proof.

As the parameters sent by the potential attacker are going to be stored in an object of type `CreateProofRequest`, it is recommended to check each of the fields of this object with regexes to ensure their format and that the content is not malicious.

Remediation Plan:

SOLVED: The function `isValidCreateProofRequest` was added to validate the `params` before using it.

3.3 (HAL-03) DEPENDENCIES SHOULD BE PINNED TO EXACT VERSIONS - LOW

Description:

The application contains some external dependencies, some of which are not pinned to an exact version but set to a compatible version (^x.x.x). This can potentially enable dependency attacks, as observed with the event-stream package with the Copay Bitcoin Wallet.

Details:

Main package.json:

Listing 6

```
"devDependencies": {
    "@metamask/eslint-config": "^10.0.0",
    "@metamask/eslint-config-jest": "^10.0.0",
    "@metamask/eslint-config-nodejs": "^10.0.0",
    "@metamask/eslint-config-typescript": "^10.0.0",
    "@typescript-eslint/eslint-plugin": "^5.33.0",
    "@typescript-eslint/parser": "^5.33.0",
    "eslint": "^8.21.0",
    "eslint-config-prettier": "^8.1.0",
    "eslint-plugin-import": "^2.26.0",
    "eslint-plugin-jest": "^26.8.2",
    "eslint-plugin-jsdoc": "^39.2.9",
    "eslint-plugin-node": "^11.1.0",
    "eslint-plugin-prettier": "^4.2.1",
    "patch-package": "^6.5.1",
    "prettier": "^2.2.1",
    "prettier-plugin-packagejson": "^2.2.18",
    "typescript": "^4.7.4"
}
```

Snap package.json:

Listing 7

```
"dependencies": {
  "@ethersproject/abi": "^5.0.0",
  "@ethersproject/bytes": "^5.7.0",
  "@ethersproject/keccak256": "^5.7.0",
  "@ethersproject/providers": "^5.7.2",
  "@iden3/js-crypto": "^1.0.0-beta.1",
  "@iden3/js-iden3-core": "^1.0.0-beta.2",
  "@iden3/js-jsonld-merklization": "^1.0.0-beta.14",
  "@iden3/js-jwz": "^1.0.0-beta.2",
  "@iden3/js-merkletree": "^1.0.0-beta.1",
  "@metamask/snaps-jest": "^0.35.2-flask.1",
  "@metamask/snaps-types": "^0.32.2",
  "@metamask/snaps-ui": "^0.32.2",
  "buffer": "^6.0.3",
  "ethers": "^5.7.2",
  "intl": "^1.2.5",
  "uuid": "^9.0.0"
},
"devDependencies": {
  "@jest/globals": "^29.5.0",
  "@lavamoat/allow-scripts": "^2.0.3",
  "@metamask/auto-changelog": "^2.6.0",
  "@metamask/eslint-config": "^10.0.0",
  "@metamask/eslint-config-jest": "^10.0.0",
  "@metamask/eslint-config-nodejs": "^10.0.0",
  "@metamask/eslint-config-typescript": "^10.0.0",
  "@metamask/snaps-cli": "^0.32.2",
  "@types/intl": "^1.2.0",
  "@types/uuid": "^9.0.2",
  "@typescript-eslint/eslint-plugin": "^5.33.0",
  "@typescript-eslint/parser": "^5.33.0",
  "esbuild": "^0.17.19",
  "eslint": "^8.21.0",
  "eslint-config-prettier": "^8.1.0",
  "eslint-plugin-import": "^2.26.0",
  "eslint-plugin-jest": "^26.8.2",
  "eslint-plugin-jsdoc": "^39.2.9",
  "eslint-plugin-node": "^11.1.0",
  "eslint-plugin-prettier": "^4.2.1",
  "jest": "^29.5.0",
  "node-stdlib-browser": "^1.2.0",
  "nodemon": "2.0.20",
  "prettier": "^2.2.1",
```

```
    "prettier-plugin-packagejson": "^2.2.11",
    "rimraf": "^3.0.2",
    "ts-jest": "^29.1.0",
    "typescript": "^4.7.4"
  }
```

Site package.json:

Listing 8

```
"dependencies": {
    "@metamask/providers": "^9.0.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-is": "^18.2.0",
    "styled-components": "5.3.3"
  },
  "devDependencies": {
    "@metamask/eslint-config": "^10.0.0",
    "@metamask/eslint-config-jest": "^10.0.0",
    "@metamask/eslint-config-nodejs": "^10.0.0",
    "@metamask/eslint-config-typescript": "^10.0.0",
    "@svgr/webpack": "^6.4.0",
    "@testing-library/dom": "^8.17.1",
    "@testing-library/jest-dom": "^5.16.4",
    "@testing-library/react": "^13.3.0",
    "@testing-library/user-event": "^13.5.0",
    "@types/jest": "^27.5.2",
    "@types/react": "^18.0.15",
    "@types/react-dom": "^18.0.6",
    "@types/styled-components": "^5.1.25",
    "@typescript-eslint/eslint-plugin": "^5.33.0",
    "@typescript-eslint/parser": "^5.33.0",
    "cross-env": "^7.0.3",
    "eslint": "^8.21.0",
    "eslint-config-prettier": "^8.1.0",
    "eslint-plugin-import": "^2.26.0",
    "eslint-plugin-jest": "^26.8.2",
    "eslint-plugin-jsdoc": "^39.2.9",
    "eslint-plugin-node": "^11.1.0",
    "eslint-plugin-prettier": "^4.2.1",
    "gatsby": "^4.24.4",
    "gatsby-plugin-manifest": "^4.24.0",
```

```
34      "gatsby-plugin-styled-components": "^5.24.0",
35      "gatsby-plugin-svgr": "^3.0.0-beta.0",
36      "prettier": "^2.2.1",
37      "prettier-plugin-packagejson": "^2.2.18",
38      "rimraf": "^3.0.2",
39      "typescript": "^4.7.4"
40    }
```

Risk Level:

Likelihood - 2
Impact - 2

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Recommendation:

Pinning dependencies to an exact version (=x.x.x) can reduce the possibility of inadvertently introducing a malicious version of a dependency in the future.

Remediation Plan:

SOLVED: All the dependencies are now pinned to exact versions.

3.4 (HAL-04) RESTRICT SITES - INFORMATIONAL

Description:

Metamask snap applications are protected by default by Metamask, which will control which origins can communicate with each installed snap.

However, with some social engineering, an attacker could manage to convince a user to provide access to a malicious origin, so it can manage to call the Rarimo's Snap App endpoints.

This could be made impossible if the snap could only allow verified origins to communicate with it.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

If possible, only allow to communicate with the snap web origins that are verified and specified in a white-list. If Rarimo does not expect third-parties websites to be communicating with the snap, only the known Rarimo websites using the snap should be allowed.

Remediation Plan:

ACKNOWLEDGED: The Rarimo team considered that third-party websites should be able to use their snap app and therefore will not be implementing this protection.

THANK YOU FOR CHOOSING

// HALBORN